第3章

総当たり方式による エラー訂正回路の設計

——Design Wave設計コンテスト 2006 Student部門 第1位

チーム マッハ255（若林秀明）

Design Wave設計コンテスト2006のStudent部門 第1位の設計を紹介する．最初，筆者はエラー数の削減を目ざしていたが，なかなか良いアルゴリズムが見つからなかった．そこで，面積と速度の最適化へと方針転換しようとしたときに理想的な信号と総当たりで比較し，もっとも近いものを選ぶ方式（最尤(さいゆう)復号法）を思いついた．回路設計では，はじめに最尤復号法のアルゴリズムどおりの設計を行い，比較的大きなブロックを削減していった．なお，本稿で紹介した回路の設計データは本誌のWebサイト（http://www.cqpub.co.jp/dwm/）からダウンロードできる．（編集部）

私が所属している研究室では，ハードウェアを設計して数値シミュレーションを高速化する研究などを行っています．研究の中では，HDLを用いて論理回路を設計し，FPGAに実装して動作させることもあります．

研究室では，2年前からこの設計コンテストに参加して，回路設計の勉強と力試しをしてきました．今年もコンテストを通して回路設計について学びたいと思いました．そして，何よりも1次審査を通過して沖縄での発表会を楽しみたいと思いました．そこで，課題のエラー訂正回路の設計にチャレンジすることにしました．

1．復号方式を考える

まず，設計仕様書[1]に示されていた繰り返し型デコーダ（エラー訂正回路）をそのままVHDLで記述してみました．符号の復号についてはなじみがなかったので，はじめは設計仕様書の内容を難しく感じました．しかし，「仕様書の回路例のとおりに作って実際に動かしてみれば，課題の理解が深まるだろう」と思い，設計を開始したのです．設計仕様書どおりのエラー訂正回路を記述し，シミュレーションしてみると，8クロックごとに復号結果が出力されていることが確認できました．

次に，この復号結果が正しいかどうかを検証するため，設計仕様書と同じ方法で復号を行うプログラムをC言語で作成しました．そして，設計した回路とプログラムで復号できたビット数が同じになり，正しく回路を設計できたことを確認しました．

その後，この回路をどのようにくふうしていくか考えました．良いエラー訂正回路の条件としては，コンパクトで，高速に動作し，エラー訂正能力が高いといったことなどが挙げられます．この中で，エラー訂正能力をより向上させるのは難しいと思いました．復号アルゴリズムの根本的なところを改良する必要があるからです．その一方で，今回のコンテストではだれも作らないようなおもしろい回路を作ってほかの人を驚かせてみたいとも思っていました．そこで，今回はあえてエラー訂正能力の向上を目ざすことにしました．

● エラー数の削減を目ざす

エラー数を減らす方法として，まず，外部値を計算するときに近似式を使わないことを考えました．設計仕様書の回路では，外部値の計算において，

KeyWord 最尤復号法，エラー訂正，2次元積符号，BPSK信号，確率，2乗器，アキュムレータ，比較回路，FPGA

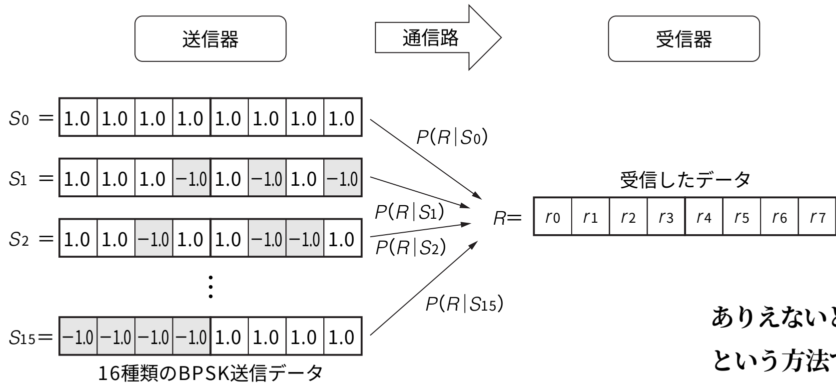


図1
復号アルゴリズムの概要
情報ビットは4ビットのため，送信器から送出される信号は16通り．そこで受信したデータにいちばん似ているものが送信されたとみなして復号する．

$$\ln\left(\frac{1+\exp(A+B)}{\exp(A)+\exp(B)}\right) \approx \mathrm{sgn}(A\cdot B)\min(|A|,|B|) \quad \cdots(1)$$

の近似を行っています．そこで，この近似を行わずに外部値を計算するようにC言語のプログラムを変更してみました．しかし，変更後も，課題の$S/N=0$dBのデータを復号したときのエラー数は700前後となり，変更前の735とそれほど変わらないことがわかりました．

また，繰り返し回数をより多くすることでエラー数が減るかどうかも試してみました．しかし，この場合もエラー数は700前後となり，思ったほどエラー数が減らないことがわかりました．このときは，がんばればエラー数が300ぐらいまで減らせるのではないか，と何の根拠もなしに思っていたのです．

その後，思いつくままにいろいろな方法を試してみました．例えば，行パリティと列パリティを使う順番を変えたり，0に近い受信値を強制的に0にして復号してみました．さらに，繰り返すごとに情報ビットのエラーが訂正されていくのと同じように，パリティの受信値も繰り返すごとに更新できないかと考えてみました．しかし，やはりエラー数は大きく減らせませんでした．図書館で符号理論の本を探してみましたが，今回のような2次元積符号の復号について詳しく解説している本は見つけられませんでした．

● 行き詰まりつつあったときにひらめく

どうやってもエラー数を減らすことができないので，行き詰まったような感じになりました．今回のコンテストは，エラー数を保ったまま，いかにして小さく高速な回路を作るかの勝負になりそうだな，と感じ始めていました．

そんな中，ふと，あるビットを‘0’か‘1’かどちらかに仮定して復号を行い，パリティに矛盾があったり確率的にありえないところがあったら最初の仮定がまちがっていた，という方法で復号できるのではないかと思いました．さらに，どうせなら送信情報を“0000”から“1111”までのすべての場合について仮定してみて，いちばん良く当てはまるものに復号すればよいのではないかと思いました．

この方法でほんとうに復号できるかどうか，最初はわかりませんでした．しかしその後，設計仕様書に記載されている式を参考にしてよく考えてみると，どうやら復号できそうだとわかりました．このアイデアでいこう，そう思い，さっそく回路設計に取りかかりました．

2．復号アルゴリズム

図1に，今回提案する復号アルゴリズムの概要を示します．今回復号する2次元積符号では，符号内の情報ビットは4ビットなので，送信器から送出されるBPSK（binary phase shift keying）信号系列は$2^4=16$通りだけです．この16通りのBPSK信号系列を，

$$S_j=(s_{j0},\ s_{j1},\ \cdots,\ s_{j7})\ (j=0,\ 1,\ \cdots,\ 15)$$

で表すことにします．また，受信器が受信した信号系列を，

$$R=(r_0,\ r_1,\ \cdots,\ r_7)$$

で表すことにします．

通信路では，送信された信号にノイズが加えられます．そのため，受信値は送信値とは若干異なる値になります．しかし，元の送信データはS_0～S_{15}の16個のうちのどれかであるので，受信器側では，S_0～S_{15}のうちRにいちばん似ているものが送信されたとみなして復号することにします．このとき，パリティの行方向や列方向といったことは考えずに，単純に8個の受信値がどのBPSK送信系列と似ているかを考えて復号します．

この復号法は，確率の考えかたを使って次のように述べることもできます．まず，16個すべてのS_jに対して，S_jを

送信したときにRを受信する条件付き確率$P(R\mid S_j)$の値を計算します．そして，もっとも$P(R\mid S_j)$が大きくなるS_jを送信されたデータとみなして復号します．

この復号法は，符号理論の分野では「最尤(さいゆう)復号法」と呼ばれています．送信情報に偏りがない場合，この復号法はもっとも誤り率が小さくなる復号法であることが知られています[(2)]．なお，複数のS_jについての確率がまったく同じでかつ最大になった場合には，そのうちのどれかに復号することになりますが，ここでどれを選ぶかによって実際のエラー数にはわずかな差が出ます．

● 復号アルゴリズムの定式化

復号結果をdとすると，dは次のように求めることができます．

$$d = \underset{j}{\operatorname{argmax}}\, P\left(R \mid S_j\right) \quad \cdots\cdots (2)$$

ここで，argmax $f(x)$は$f(x)$を最大にするxを表します．

復号に必要なものは$P(R\mid S_j)$の大小関係であり$P(R\mid S_j)$の絶対的な大きさは必要ありません．そこで，次のように対数をとることができます．

$$d = \underset{j}{\operatorname{argmax}}\, \ln\left\{P\left(R \mid S_j\right)\right\} \quad \cdots\cdots (3)$$

さらに，Rを受信する確率は，初めにr_0を受信し，次にr_1を受信し，以下同様にr_7まで受信する確率であるので，1ビットごとの確率の積で表すことができます．

$$d = \underset{j}{\operatorname{argmax}}\, \ln\left\{\prod_{k=0}^{7} P\left(r_k \mid s_{jk}\right)\right\} \quad \cdots\cdots (4)$$

ここで，$\prod_{k=0}^{7}$は$k=0$～7の項を掛け合わせる記号です．白色ガウス・ノイズの性質から，

$$d = \underset{j}{\operatorname{argmax}}\, \ln\left\{\prod_{k=0}^{7}\left(\frac{1}{\sqrt{2\pi}\cdot\sigma}\exp\frac{-\left(r_k - s_{jk}\right)^2}{2\sigma^2}\right)\right\} \quad \cdots (5)$$

となります．σ^2は通信路におけるノイズの分散です．

式(5)からjに関係ない項を取り除くと，

$$d = \underset{j}{\operatorname{argmax}}\, \ln\left\{\prod_{k=0}^{7}\exp\left(-\left(r_k - s_{jk}\right)^2\right)\right\} \quad \cdots\cdots (6)$$

となり，指数と対数の性質から，

$$d = \underset{j}{\operatorname{argmax}} \sum_{k=0}^{7}\left(-\left(r_k - s_{jk}\right)^2\right) \quad \cdots\cdots (7)$$

と変形できます．s_{jk}は±1.0であるので，2乗の項を展開すると，

$$\begin{aligned} d &= \underset{j}{\operatorname{argmax}} \sum_{k=0}^{7}\left(-r_k^2 + 2r_k s_{jk} - 1.0\right) \\ &= \underset{j}{\operatorname{argmax}} \sum_{k=0}^{7} r_k s_{jk} \quad \cdots\cdots (8) \end{aligned}$$

となります．さらに，下記のようなy_{jk}を使えば，

$$d = \underset{j}{\operatorname{argmin}} \sum_{k=0}^{7} r_k y_{jk},\quad y_{jk} = \begin{cases} 0\left(s_{jk} = 1.0\right) \\ 1\left(s_{jk} = -1.0\right) \end{cases} \quad \cdots\cdots (9)$$

となります．以下では$\Sigma r_k y_{jk}$をΣjと略記します．

以上のように，最終的には和の形で確率$P(R\mid S_j)$に対応する値を求めることができます．加算器と比較回路を用いて復号することができるため，このアルゴリズムはハードウェア化にも適しています．

以上より，今回設計する回路では以下のように復号を行います．

1）手順1：すべての送信情報に対して，下記のようにΣjを計算する

- 送信ビット“0000 0000”：$\Sigma 0 = 0$
- 送信ビット“0001 0101”：$\Sigma 1 = r_3 + r_5 + r_7$
- 送信ビット“0010 0110”：$\Sigma 2 = r_2 + r_5 + r_6$

　⋮

- 送信ビット“1111 0000”：$\Sigma 15 = r_0 + r_1 + r_2 + r_3$

2）手順2：最小のΣjを探して復号する

例えば，Σ2が最小となったときには“0010”に復号します．まれに，二つ以上のΣjが最小値となる場合もあります．この場合には，最小値となるjのうちの任意の一つを復号結果とします．

3．回路設計

最尤復号法のアルゴリズムをハードウェアで実現することを考えました．

図2に，最初に考えた回路のブロック図を示します．この回路は，式(7)を計算して復号を行います（この時点では，式(8)以降の変形ができることに気づいていなかった）．最初の部分で，$(r-32)^2$と$(r+32)^2$を計算し，セレクタでどちらかを選択します．アキュムレータで和を計算し，その後ろの回路で16個の値を比較して復号します．

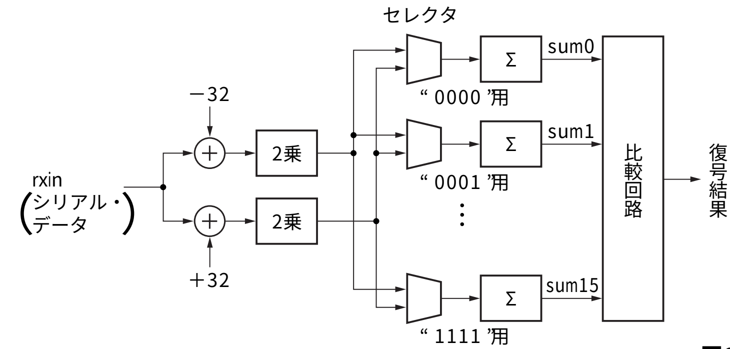


図2　最尤復号法を用いたデコーダ回路(効率化前)
式(7)を計算して復号を行う回路である．最初の部分で，$(r-32)^2$と$(r+32)^2$を計算し，セレクタでどちらかを選択する．アキュムレータで和を計算し，その後ろの回路で16個の値を比較して復号する．

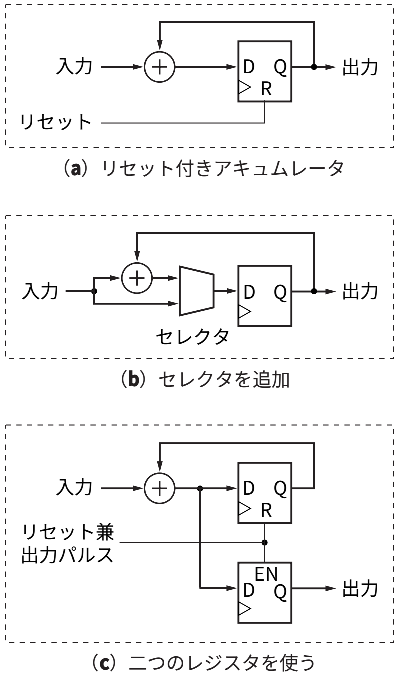


図3
アキュムレータのくふう
(a)はもっとも簡単なリセット付きアキュムレータの回路．(b)のようにすると，リセット信号と同じクロックに入力されたデータを加算することができる．(c)は(b)よりも動作速度を上げるために二つのレジスタを使う構成．

この回路には繰り返し型エラー訂正回路のようなループ状のパスがないため，パイプライン処理を行えば最大動作周波数を高くできそうだという期待が持てました．

● 2乗器の設計

最初は，式(7)の$(r\pm32)^2$の値を計算するために，符号付きの2乗器を設計していました．VHDLの演算子*を使わずに，自分で論理を考えてパイプライン2乗器を設計することによって，回路面積と動作速度を改善できました．最終的にはこのブロックは不要になりましたが，このブロックの設計はおもしろく勉強になるものでした．

なお，±32の値は，今回の課題においてノイズがないときのBPSK送受信値です．

● アキュムレータの設計

式(7)の和を求めるために，アキュムレータ(累算器)を使用することにしました．**図2**のように16個のアキュムレータを並列に接続し，すべての送信情報に対する計算を同時に行います．受信信号rxinはシリアル・データなので，8クロックの間に$(r-32)^2$または$(r+32)^2$を足し合わせることによって計算します．

各アキュムレータは，受信値8個ごとにリセットを行う必要があります．ここで，加算とリセットを同時に行うために回路をくふうしました．

もっとも簡単なリセット付きアキュムレータの回路は**図3(a)**の回路です．しかし，この回路ではリセット信号と同じクロックに入力されたデータを加算することができません．**図3(b)**のような回路にすればこの問題を解決できますが，1クロックの間にデータがセレクタと加算器の両方を通るため，動作速度が遅くなることが考えられます．そこで，今回は**図3(c)**のように二つのレジスタを使うことにしました．この回路は1クロックの間に行う処理が少ないので，高速に動作します．出力側のレジスタは，イネーブル信号を使って値を8クロックの間保持するようにしました．これによって，後段の回路をうまく作ることができました．

● 比較回路の設計

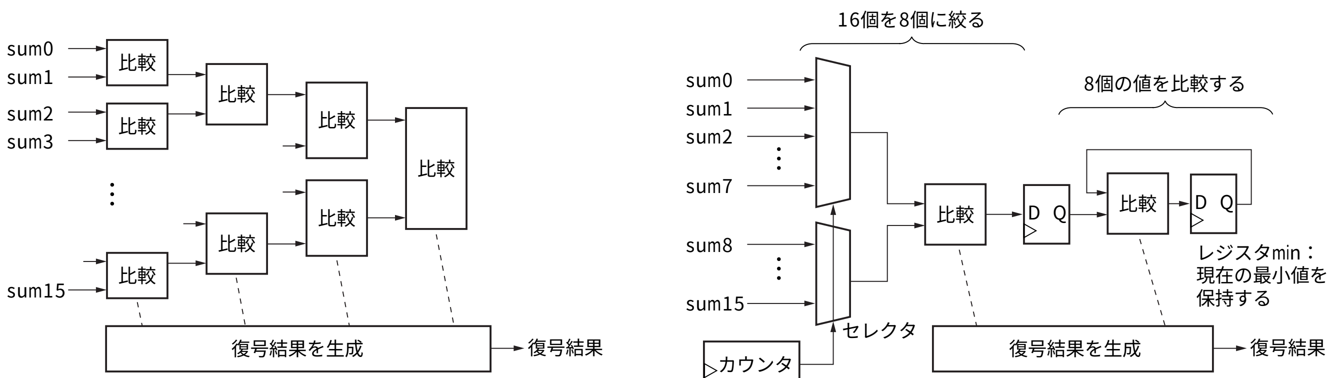
16個の値の大きさを比較して復号を行う回路のブロック図を**図4**に示します．初めは15個の比較器を用いて，**図4(a)**のような形の回路を設計しましたが，それでは回路規模が大きくなってしまうので，**図4(b)**のように同じ比較器を何回も使い回すことによって回路規模を縮小させました．

8クロックごとに16個の値を処理するために，二つのステージを用意しました．ステージ1では16個の値を8個に絞り，ステージ2では8個の値の中で最小となるものを探します．

1) ステージ1

前段から送られてくる16個の値は，8クロックの間，一定です．そこで，まず**図4**のように，セレクタを用いて前

3



（a）15個の比較器を用いた回路　（b）比較器を使い回す回路

図4　比較回路

(a)は15個の比較器を用いたもの．(b)は比較器を何回も使い回すことによって回路規模を縮小させたもの．

段の値を一つずつ取り出します．そして，count番目の値と(count＋8)番目の値を比較して，小さいほうをステージ2に送ります．ここでcountは0～7を数えるカウンタの出力です．

このような回路構成にすると，一つの比較器で，8クロックの間に8回の比較を行うことができ，効率的です．

2）ステージ2

ステージ2では，シリアル・データとして次々と送られてくる8個の値を比較します．

まず，現在の最小値を格納しておくためのレジスタminを用意します．このレジスタは，比較を開始する前に，2の補数で表現できる最大値で初期化しておきます．

ステージ1からデータが送られてきたら，送られてきた値とレジスタminに格納されている値を比較します．送られてきた値のほうが小さい場合には，レジスタminの値を更新します．

この二つのステージの比較結果から復号結果を生成して，出力します．

● 式の変形と2乗器の廃止

ここまでのくふうで，だいぶオリジナリティが出せたと思っていました．しかし，インターネットで「最尤復号法」を検索してみると，思ったよりも多くの件数がヒットしました．また，この復号法を用いてさまざまな研究を行っている方もいました．そこで，もしかしたらほかのチームもこの方法で応募してくるかもしれない，と思い，何かほかの人と重ならないようなアイデアが出せないかと考えました．

そのような中で，2乗器は比較的大きな回路面積を占めていました．そこで，$(r-32)^2$と$(r+32)^2$を計算する2乗器を一つにまとめられないかと考えました．

両方から同じ値を減算しても復号結果には影響がないので，ためしに両方から$(r-32)^2$を引いてみました．すると片方は0になって，もう片方は2乗の項が消え，計算式が簡単になりました．これに気づいたときは驚きました．

その後，データパスのビット幅などを再検討して，回路全体を再設計しました．

● 最終的な回路

最終的には，回路は**図5**のようになりました．初期の回路と比べて，±32の加算器と2乗器がなくなりました．また，データパスのビット幅も減り，回路規模を大幅に減らすことができました．

アキュムレータでは，入力イネーブル信号を利用して必要な受信値のみを加算するようにしました．この入力イネーブル信号は式(9)のy_{jk}に対応しています．アキュムレータをリセットすると同時に送られてくるr_7については，加算するかしないかを決めるために，**図6**のように2種類の回路を用いることにしました．アキュムレータAはr_7を加算し，アキュムレータBはr_7を加算しません．

16個のΣjを8個に絞る部分では，(Σ2，Σ3，Σ4，Σ5，Σ6，Σ7，Σ0，Σ1)および(Σ10，Σ11，Σ12，Σ13，Σ14，Σ15，Σ8，Σ9)の順で値を取り出していますが，これはタイミングのつごうです．この順にすると，復号結果を生成するときにカウンタの値をそのまま結果として用いることができます．

タイミングを制御するために，0～7を数えるカウンタ

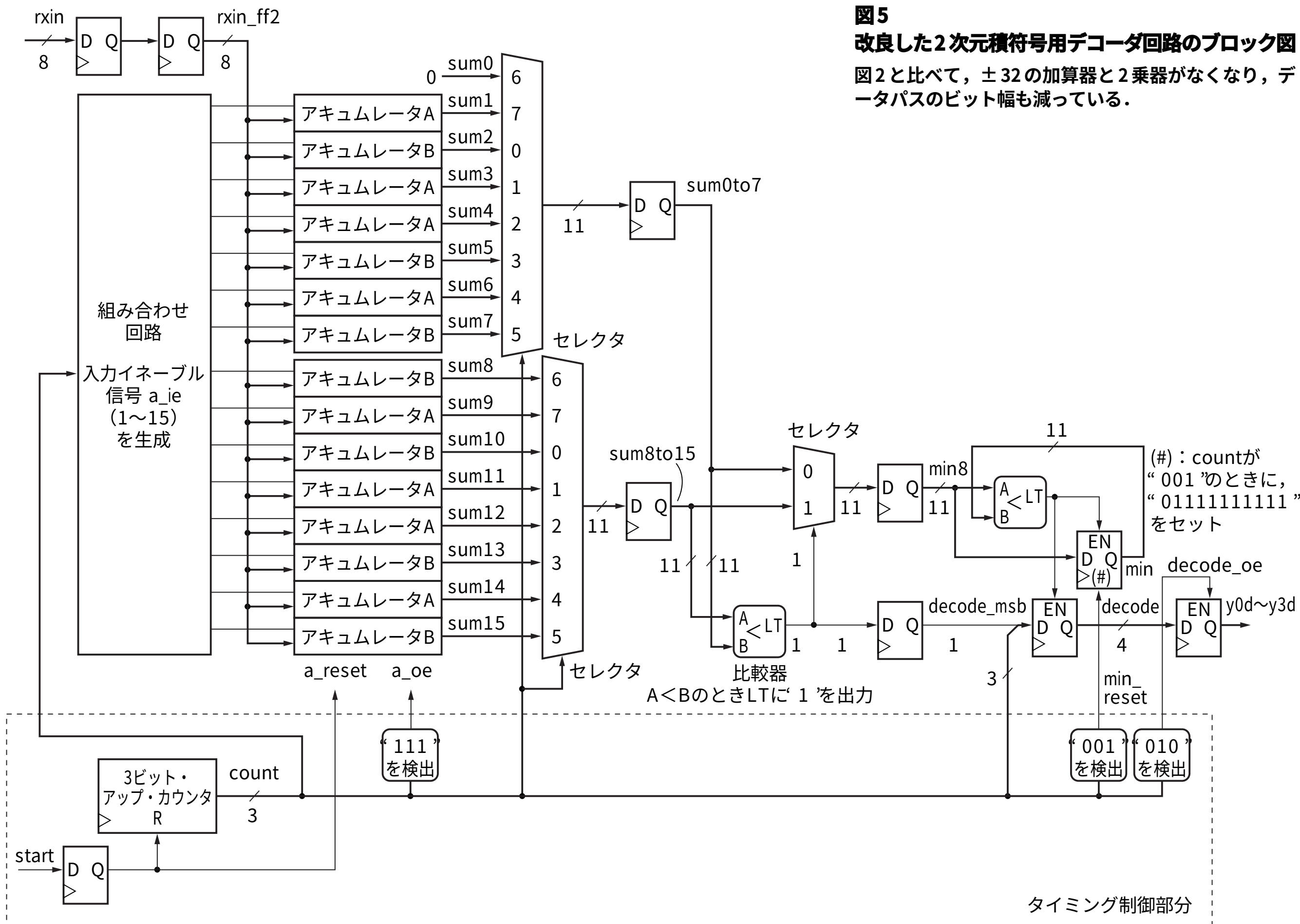


図5
改良した2次元積符号用デコーダ回路のブロック図
図2と比べて，±32の加算器と2乗器がなくなり，データパスのビット幅も減っている．

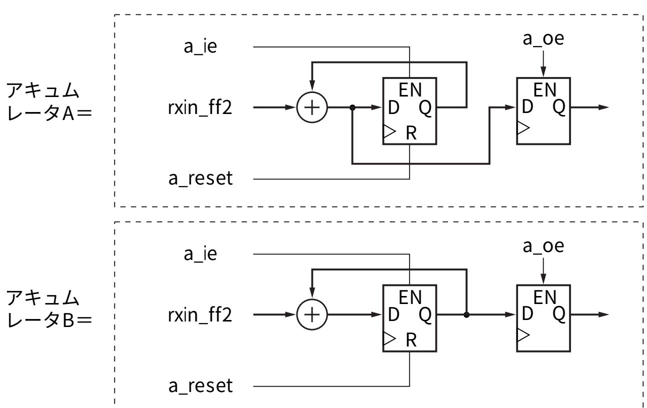


図6　2種類のアキュムレータ
アキュムレータAはr_7を加算し，アキュムレータBはr_7を加算しない．

リスト1　入力イネーブル信号を生成する組み合わせ回路のVHDLソース・コード
r_7の処理は別の部分で行っているため，r_7に対する入力イネーブル信号はドント・ケアである．そこで，ここでは不定値‘X’を入力することによって論理の縮小を図っている．

```
   -- (for sum1～15)
 a_ie <= "000000011111111" when (count = "000") else
         "000111100001111" when (count = "001") else
         "011001100110011" when (count = "010") else
         "101010101010101" when (count = "011") else
         "000111111110000" when (count = "100") else
         "110011001100110" when (count = "101") else
         "011001111001100" when (count = "110") else
         "XXXXXXXXXXXXXXX"; -- don't care
```

を用意しました．カウンタの値からイネーブル信号，リセット信号などを生成して，各部の制御を行っています(**リスト1**)．

この回路のタイミング図を**図7**に示します．復号結果は，回路にr_7を入力してから14クロック後に出力されます．入力データは設計仕様書のタイミング図と同じように連続的に送られてくるという前提で回路を設計してあります．

4．回路の評価

今回は，FPGAに実装することを想定して，回路の論理合成と評価を行いました．論理合成には米国Xilinx社の

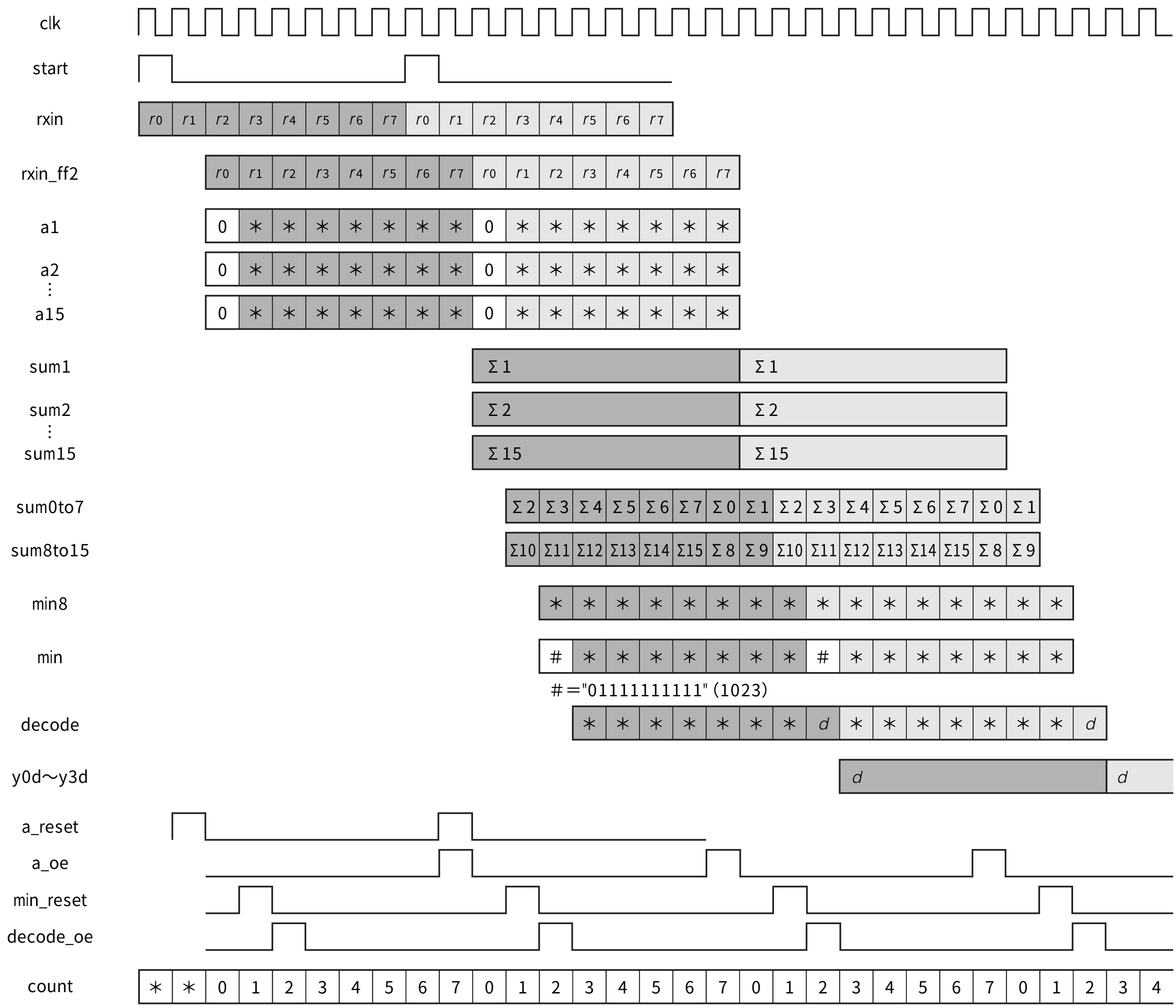


図7　設計した回路のタイミング図
復号結果は，回路にr_7を入力してから14クロック後に出力される．

ISE 6.3.03iを，論理シミュレーションには米国Mentor Graphics社のModelSim SE 5.7dを使用しました．そして論理合成時のターゲット・デバイスには，Xilinx社のVirtexシリーズの一つであるXCV300-4PQ240を選択しました．ただし，今回は論理合成と論理シミュレーションのみを行い，実際にFPGAボードに実装しての評価・検証は行いませんでした．

表1に，設計したエラー訂正回路の使用スライス数，動作速度，エラー数を示します．ここで，スライスとは論理回路を実現するためのFPGA内のブロックです．今回想定するターゲット・デバイスでは，各スライスには2個の4入力ルックアップ・テーブル，2個のDフリップフロップなどが含まれています．

設計仕様書の回路と比べて，今回設計した回路では最大動作周波数が約4.1倍になりました．これは，今回設計した回路には繰り返し型エラー訂正回路のような長いパスがないためです．また，エラー訂正能力も繰り返し型の回路より良くなりました．これは，復号アルゴリズムの性質によるものであり，精度を落として計算している部分もないためです．それにもかかわらず，使用スライス数は設計仕様書の回路例と比べて約3/4と少なくなりました．

図8に，回路の動作シミュレーション波形を示します．図より，r_7を入力してから14クロック後に復号結果が出力されていることを確認できます．

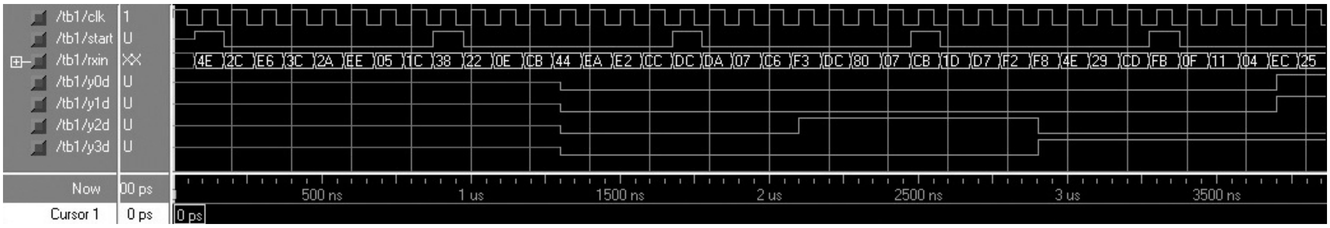


図8　設計した回路のシミュレーション波形
r_7を入力してから14クロック後に復号結果が出力されていることを確認できる．

● 試さなかったほかのアイデア

今回は，実装するデバイスの特徴などはとくに気にせずに設計しました．ターゲット・デバイスを特定して，そのデバイスに合わせた設計を行えば，より良い性能を実現できたかもしれません．

また，今回は直せるエラーはすべて直せるように設計しました．しかし，わずかなエラー数の増加を許容できるのであれば，もっと回路規模を小さくできたと思います．例えば，今回はデータパスのビット幅をオーバフローが起きないように11ビットにして設計しましたが，8個の受信値がすべて＋127のような大きな値になる確率は非常に低いと考えられるので，MSB側のビットは何ビットか削ってもいいように思いました．

さらに，回路規模はより大きくなりそうですが，パイプラインの段数を増やせば，動作速度をさらに上げることができたかもしれません．

● コンテストに参加して

今回のコンテストでは，回路設計を通して一種の喜びを感じることができました．復号方法を考えていく中で，ふと，最尤復号法が使えると気づいたときです．その後，式を変形することによって回路を簡略化できるとひらめいたときは，さらに喜びが大きくなりました．

沖縄で行われた発表会では，どのチームの発表もユニークで，楽しいものでした．とくに，今回の課題ではどのぐらいエラーが直っているのかを画像や音声で直感的にわかればよいと思っていましたが，映像のノイズ除去という方法を用いてエラー訂正の効果をプレゼンテーションしたチームもあり，とても興味深いものでした．また，実際に動くハードウェアを会場に持ってきて音声を流したり，戦車の模型を動かしたりするチームもありました．繰り返し型の回路でもエラー数を減らしたり回路面積を小さくしたりするさまざまなくふうがあり，勉強になりました．

表1　デコーダの性能比較
設計仕様書の回路と比べて，今回設計した回路では最大動作周波数が約4.1倍になった．

		設計仕様書の回路	最尤復号法を用いた回路（最適化後）
使用スライス数		361	265
最大動作周波数（MHz）		25.320	104.734
10,000ビット当たりのエラー数	S/N=9dB	0	0
	S/N=6dB	2	2
	S/N=3dB	141	126
	S/N=0dB	735	692

コンテスト後，東京大学のチームの方から，さらなる最適化のアイデアをいただきました．8個の受信値をアキュムレータで加減算する部分において，送信データ“0000”と“1111”，“0001”と“1110”，以下同様に“0111”と“1000”の各組の送信データについて計算する加減算器を共有することで，さらに回路規模を縮小するというものでした．加減算器の数を16個から8個に減らすことによって，使用スライス数を150程度まで減らすことができます．このことを教えられたときには驚きました．自分の気づかないところで，まだまだくふうの余地があるものですね．

今回のコンテストでは，自分では思いもつかなかった多くのアイデアを知ることができ，たいへん勉強になりました．今後，回路設計を行っていくときにも，この経験を生かしていきたいと思います．

参考・引用*文献
(1) 和田知久；2次元積符号用繰り返し型デコーダ設計仕様書，pp.131-140，Design Wave Magazine，2005年11月号．
(2) 宮川洋，岩垂好裕，今井秀樹；符号理論，pp.26-28，電気情報通信学会，2001年．

わかばやし・ひであき
千葉大学大学院 自然科学研究科 電子情報システム専攻 修士2年（当時は1年）

3